Note this PDF was generated from a website available at (http://bit.ly/RESTfulAPI_ck)

Go there for all the live linked goodness.

Purpose

Can you teach a computer to recognize the brand logos in an unlabeled feed of images from Instagram?

In a previous project, I created a Convolutional Neural Network (CNN) that can identify brand logos (specifically Nike and Altra) in untagged/ unlabeled photos from a social media feed. The model I used implemented a previously trained network and transfer learning.

For this project I wanted to build a REST API that allowed me to send requests to that neural network and store its results in JSON.

Fortunately, I also had the opportunity to develop a solution for the GapJumpers Challenge available here.

The challenge specifies the following for a successful submission:

- Write a simple Rest API with PHP, Node.js, Ruby, Python or Go.
- Your API must have:
  - at least three endpoints
  - all endpoints must be linked in some way
- Pay careful attention to REST best practices
- Describe the application in detail in a design document including why you chose the approach you did.
- Include tests that validate success.

Deliverables

- Using Python and Flask, I created this website- a single page that allows for retrieval of data from my endpoints. The 3 endpoints are described in detail in the API Documentation section below.
- In order to share my application and provide links where it can be tested, links and curl embeds are included throughout this document that allow for interacting with the API directly. The API is currently live at http://10la.pythonanywhere.com/img/api/v1.0/images and all references to a [hostname] in this documentation refers to `http://10la.pythonanywhere.com`. The application, it's unit tests, and all supporting documentation can be found at the project's GitHub.
- This webpage represents the design document. In particular the sections dedicated to the REST Architecture and API Documentation discuss the design process in detail.
- Unit tests, including those that validate success, are available in a number of ways. Basic functionality

can be verified by executing each curl command in the order it appears in the the API Documentation section. A copy of the output of the unit test application is included in the Unit Tests section below. Finally, the test application can be downloaded from GitHub and run locally.

REST Architecture

Representational state transfer (REST) was originally specified in the 5th chapter of Roy Fielding's PhD thesis "Architectural Styles and the Design of Network-based Software Architectures" which specifies 6 constraints for the REST architecture.

1. Client-Server - API is separated from a client, although this could represent different processes within a single computer. The objective is to allow for scalability- mulitple clients and multiple types of clients can thus connect with a single API.
2. Stateless - No cookies and no sessions. Clients must authenticate upon every request.
3. Cache - Server should provide caching directives so that in the event that a similar request arrives multiple times, a response can be provided without requiring a repeated request to the server. There are a variety of caches, including inside of the browser.
4. Layered System - Both the client and the API may or may not be communicating directly with each the other. Likely there's a layer in between (such as a load balancer). This prevents a server from identifying requests by the client. In the case of a load-balancer, a server may receive *ALL* of its requests from a single client. The benefit of this is that in the event that a load balancer is placed in between the server and the client, you now have the ability to scale to a huge number of clients and servers without requiring fundamental changes.
5. Code-On-Demand - In practice this is an optional REST principle. Clients can receive executable code to run as a response to a request. How would an API know what types of code that a client could run?

6. Uniform Interface -

    i. Identification of resources - Resources represent every entity in the domain of the application. Each resource gets its own unique identifier URI such as: `/img/api/v1.0/images/3`. Collections of resources can also have identifiers such as `/img/api/v1.0/images`.
    ii. Resource Representations - Clients only access representations of a resource, never the resource itself. Additionally, clients only operate on representations of resources. The server may provide a number of different content types, like JSON or XML.
    iii. Self-Descriptive Message - Clients send and receive HTTP. Request method defines the operation; target is represented in the URI of the request, headers provide authentication credentials and content types, the representation of the resource is in the body, and the result of an operation is in the response status code.
    iv. HATEOAS Hypermedia As The Engine Of Application State - Clients don't know any resource URIs except for the root of the API and everything else can be discovered through links in resource representations.

Because REST defines an architectural style and is not a specification, there is room for interpretation (or

arguments on StackOverflow as the case may be).

My project's interpretation of REST constraints

1. Client-Server - Yes, the API is hosted on a server both physically and logically separated from the clients.
2. Stateless - Yes, outside of the root URI for the API clients must authenticate upon every request. Although as listed in the improvements section, adding rate limiting would potentially remove this constraint. In a stateless system how would you identify an un-authenticated user in order to limit their number of requests?
3. Cache - Yes, this is being met when the API is consumed in a browser. However, as discussed in the improvements section, adding etags would improve the existing solution.
4. Layered System - Yes, Similar to Client-Server above, all requests are authenticated separately, and there are no systems in place that would attempt to identify a client merely by its requests. I use HTTP Basic Auth and discuss some ways this could be improved in the improvements section.
5. Code-On-Demand - No, I do not meet this optional requirement of REST. One potential application that I could see this constraint being used for is to download and execute JavaScript in the browser.

6. Uniform Interface -

    i. Identification of resources - Yes, each resource gets its own unique identifier URI such as: `/img/api/v1.0/images/3` and each collections of resources has an udentifier `/img/api/v1.0/images`. I also include versioning information in this category. And while there is some argument about *where* to place the version information for your API, there is none as to *if* you should version it. I place versioning info in the URI. The other alternative is in the header.
    ii. Resource Representations - Yes, clients can only operate on representations of resources. In this case, the server only provides JSON content. I use JSON because it's ubiquitous, human readable, and easy to work with.

    iii. Self-Descriptive Message - Yes, the target is represented by the URI of the request, headers provide authentication credentials and content types, the representation of the resource is in the body, and the result of an operation is in the response status code. Request methods define the operation being performed as follows:

        - GET requests get data
        - POST requests create new data
        - PUT requests update existing data - although there are differing opinions that says you should use POST for everything.
        - DELETE requests delete data

    iv. HATEOAS Hypermedia As The Engine Of Application State - Yes, Every resource request returns a URI property (except DELETE) that is the full URI of where a resource resides. Potential improvements for this feature are discussed below.

Overall I would say that my implementation is a level 2/3 on the Richardson REST Maturity Model.

API Documentation

Below I describe and give examples for all available API resources. By following along and executing the embedded curl commands in order one can effectively test the API.

Endpoint 1. Images File Store

The first endpoint provides a means to store and retrieve information that relates to images. It handles all of the portions of our API related to creating, updating, retrieving and deleting individual image records as well as lists of images.

HTTP Method	URI	Action
GET	[hostname]/img/api/v1.0/imgs	Retrieve list of images
POST	[hostname]/img/api/v1.0/imgs	Create new image
GET	[hostname]/img/api/v1.0/imgs/[img_id]	Retrieve an individual image
PUT	[hostname]/img/api/v1.0/imgs/[img_id]	Update an existing image
DELETE	[hostname]/img/api/v1.0/imgs/[img_id]	Delete an existing image

Each image record will have a number of different fields of JSON data:

- `id` and `uri` : Unique identifier for images. Although id integers are returned as a full URI that controls the image. This means a client does not need to construct a URI from information it receives from the API but rather receives a usable full path. An example- `'id'=1` and `'uri'="http://10la.pythonanywhere.com/img/api/v1.0/images/1"` .
- `title` : A short description of the image.
- `url` : A location of the image as stored on AWS S3, such as `http://imgdirect.s3-website-us-west-2.amazonaws.com/nike.jpg`

- `results` : The probabilities and labels for each potential class an image can belong to. The below sample shows that there is a 79% confidence that the image in question is a Nike.

```
    "results": {
        "results_name_1": "nike",
        "results_score_1": "\"0.7914\""
    }
```

- `resize` : Boolean used for determining if the image size has been processed to the appropriate size.
- `size` : Height and width in pixels of an image as returned from the `resize` resource.

GET [hostname]/img/api/v1.0/imgs

```
```

Results:

```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:14:44 GMT
Content-Type: application/json
Content-Length: 509
Connection: keep-alive
Vary: Accept-Encoding
X-Clacks-Overhead: GNU Terry Pratchett

{
  "images": [
    {
      "title": "Nikes",
      "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/nike.jpg",
      "uri": "http://10la.pythonanywhere.com/img/api/v1.0/images/1",
      "results": "",
      "resize": false,
      "size": ""
    },
    {
      "title": "Altra",
      "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/altra.jpg",
      "uri": "http://10la.pythonanywhere.com/img/api/v1.0/images/2",
      "results": "",
      "resize": false,
      "size": ""
    }
  ]
}
```

POST [hostname]/img/api/v1.0/imgs

Results:

```
HTTP/1.1 201 CREATED
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:16:34 GMT
Content-Type: application/json
Content-Length: 237
Connection: keep-alive
X-Clacks-Overhead: GNU Terry Pratchett

{
  "image": {
    "title": "",
    "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/neither.jpg",
    "uri": "http://10la.pythonanywhere.com/img/api/v1.0/images/3",
    "results": "",
    "resize": false,
    "size": ""
  }
}
```

GET [hostname]/img/api/v1.0/imgs/[img_id]

Results:

```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:18:26 GMT
Content-Type: application/json
Content-Length: 181
Connection: keep-alive
X-Clacks-Overhead: GNU Terry Pratchett

{
  "img": {
    "title": "",
    "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/neither.jpg",
    "results": "",
    "id": 3,
    "resize": false,
```

```
    "size": ""
  }
}
```

```
        |
```

PUT [hostname]/img/api/v1.0/imgs/[img_id]

```

```

Results:

```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:58:11 GMT
Content-Type: application/json
Content-Length: 181
Connection: keep-alive
X-Clacks-Overhead: GNU Terry Pratchett

{
  "img": {
    "title": "C-ron-ron",
    "url": "https://s3-us-west-2.amazonaws.com/imgdirect/altra.jpg",
    "results": "",
    "id": 2,
    "resize": false,
    "size": ""
  }
}
```

DELETE [hostname]/img/api/v1.0/imgs/[img_id]

```

```

Results:

```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:22:52 GMT
Content-Type: application/json
Content-Length: 20
```

```
Connection: keep-alive
X-Clacks-Overhead: GNU Terry Pratchett

{
  "result": true
}
```

Endpoint 2. Image Inference

The image inference endpoint supplies a resource for interacting with the TensorFlow model I use for determining whether a given image contains and brands of interest. Results are returned as a probability for each of 3 possible outcomes- Altra, Nike, or Neither and are appended to the JSON associated with a given image along with the name for each outcome.

HTTP Method	URI	Action
PUT	[hostname]/img/api/v1.0/inference/[img_id]	Measure image dimensions and add height and width to field `"size"` in JSON

PUT [hostname]/img/api/v1.0/inference/[img_id]

```

```

Results:

```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:24:20 GMT
Content-Type: application/json
Content-Length: 458
Connection: keep-alive
Vary: Accept-Encoding
X-Clacks-Overhead: GNU Terry Pratchett

{
```

```
  "img": {
    "title": "Nikes",
    "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/nike.jpg",
    "uri": "http://10la.pythonanywhere.com/img/api/v1.0/images/1",
    "results": {
      "results_name_3": "altra",
      "results_name_2": "neither",
      "results_name_1": "nike",
      "results_score_3": "\"0.0008\"",
      "results_score_2": "\"0.2078\"",
      "results_score_1": "\"0.7914\""
    },
    "resize": false,
    "size": ""
  }
}C
```


Endpoint 3. Image Resize

The third endpoint begins the implementation of resizing images. Because I will eventually extend this API out to be able to ingest a photo, resize it, then upload it to AWS S3, this endpoint lays the groundwork for that function.

HTTP Method	URI	Action
PUT	[hostname]/img/api/v1.0/resize/[img_id]	Measure image dimensions and add height and width to field `"size"` in JSON

PUT [hostname]/img/api/v1.0/resize/[img_id]

Results:


```
HTTP/1.1 200 OK
Server: openresty/1.9.15.1
Date: Thu, 25 Aug 2016 11:25:27 GMT
Content-Type: application/json
```

```
Content-Length: 504
Connection: keep-alive
Vary: Accept-Encoding
X-Clacks-Overhead: GNU Terry Pratchett

{
  "img": {
    "title": "Altra",
    "url": "http://imgdirect.s3-website-us-west-2.amazonaws.com/altra.jpg",
    "uri": "http://10la.pythonanywhere.com/img/api/v1.0/images/2",
    "results": {
      "results_name_3": "nike",
      "results_name_2": "altra",
      "results_name_1": "neither",
      "results_score_3": "\"0.0316\"",
      "results_score_2": "\"0.2004\"",
      "results_score_1": "\"0.7680\""
    },
    "resize": false,
    "size": {
      "width": 480,
      "height": 480
    }
  }
}
```

Unit Tests

The unit tests in `test_app.py` run a number of different tests against the API.

They test:

1. A root address exists and responds appropriately
2. A nonexistent URL returns a 404 error
3. Individual images have their own URI that responds correctly
4. Bad image URLs return a 404 error
5. Creating a new image functions and returns the appropriate response
6. JSON is used throughout the requests and responses
7. Authentication is used correctly on all requests
8. The inference resource works on an image and returns the correct response
9. The resize resource functions correctly and returns the correct response
10. Deleting an image functions and returns the correct response

Below is sample output from running the unit tests from the file `test_app.py`. Note that there is a deprecation warning from TensorFlow that is expected.

```
    $ python test_app.py
    ...tensorflow/core/framework/op_def_util.cc:332] Op BatchNormWithGlobalNormalization is depre
```

```
...
----------------------------------------------------------------------
Ran 13 tests in 4.186s

OK
```

Improvements

- Better authentication beyond HTTP Basic
- HTTPS for every request
- Add a database
- Better HATEOAS compliance
- Improved caching via etags.
- Better type checking for images
- Add pagination
- Add rate limiting
- Add direct upload to AWS S3 with IAM roles

References

- Best Practices for a Pragmatic RESTful API
- Is Your REST API RESTful? - PyCon 2015
- Designing a RESTful API with Python and Flask
- The Flask Mega-Tutorial
- What Exactly is REST Programming?
- How to design a REST API
- REST API Design Guidelines